FXTH87xx1xFWUG

FXTH87xx11 and FXTH87xx12 embedded firmware user guide

Rev. 3 — 17 August 2022 User guide

Document information

Information	Content
Keywords	FXTH87xx1xFWUG, FXTH87xx11, FXTH87xx12, embedded firmware, user guide, global variables, standard format used in functions, function description
Abstract	This document describes the embedded firmware found in all derivatives of the FXTH87xx11 and FXTH87xx12 devices. The intended audience for this document is firmware architects, developers, coders and/or testers working with the FXTH87xx11 and FXTH87xx12 devices.

Revision history

Rev	Date	Description
v.3	20220817	• Global: performed minor grammatical and typographical changes throughout. • Added Document information table at the start of the document containing keywords and an abstract. In addition, the revision history was relocated from the end to the beginning of the document. The changes were made to comply with the NXP document content hierarchy for user guides and user manuals • Section 1, moved the first two paragraphs of the introduction into the abstract of the document information section. • Section 3.2.6, Table 12, revised the U8Status Value in the last row from "$00" to "$20". • Section 3.2.34, inserted "The size must be strictly greater than 0. Setting the size to 0 will cause an unexpected behavior" at the end of input parameter "UINT16". • Section 3.2.36, inserted note, "The result may include an error due to truncation in the calculation." • Section 3.2.41. inserted "The number of bytes must be strictly greater than 0. Calling the function with a number of bytes equal to 0 may cause user FLASH memory corruption." on input parameter UINT8 u8Size • Section 3.2.46, inserted note at the end of the section.
v.2.2	20170710	• The format of this data sheet has been redesigned to comply with the new identity guidelines of NXP Semiconductors. Legal texts have been adapted to the new company name where appropriate. • Changed "usec" to "µs" throughout document. • Added "If an LFR interrupt occurs while a firmware function is under execution, the LFR User Interrupt Vector will not be accessed, and the bit 2 below will be the only indication available. Users should check this bit, either prior to entering the firmware function or after the firmware function, to assure interrupts are not missed. Also, a number of firmware functions utilize the Stop1 or Stop4 modes, which disable the hardware Watch-dog block. In order to provide a back-up recovery, users should utilize either the RTI or PWU which can be programmed for interrupt if a software or firmware routine has consumed too much time. The Watch-dog is automatically restarted when the program goes back in RUN mode." to the end of the first paragraph of Section 2.1.1. • Revised the description for the field "Bit 7 - ADCERR" from "...TPMS_WIRE_AND_ADC_CHECK routine" to "...TPMS_WIRE_AND_ADC_CHECK routine, or when the measurement could not be acquired after the execution of a TPMS_READ function" in Table 3 of Section 2.2.2. • Added a fourth column identifying the section and creating hyperlinks to each section within Table 7 in Section 3.1. • Revised the title of Table 17 in Section 3.2.11 from "Valid output conditions for TPMS_READ_ ACCELERATION_X" to "Valid output conditions for TPMS_READ_ACCELERATION_Z". • Revised the item below "Returns" from "UINT8 u8WDIV: WDIV compensated value." to "UINT8 u8WDIV: WDIV compensated value, or $80 if the XTAL was not found." in Section 3.2.19. • Added two sentences, "The execution of this routine will change the contents of RFM registers. Specifically note that RF Direct Mode will be selected after its execution." to the warning Section 3.2.19. • Added three sentences, "Prior to calling this routine, the RFM must be enabled. The execution of this routine will change the contents of RFM registers. Specifically note that RF Direct Mode will be selected after its execution." to the warning in Section 3.2.20. • Changed "It requires the MCU to be configured for 4 MHz bus clock, and the RFM to be enabled but not transmitting prior to making the call." to "It requires the MCU to be configured for 4 MHz bus clock, and the RFM to be enabled but not transmitting prior to making the call." in the power management items of Section 3.2.19 and Section 3.2.20. • Added new item "When BIT5 of u8PowerManagement is set, find the best RF power setting (RFCFR2_PWR) dynamically based on voltage, temperature, and current carrier frequency in order to target 3 dBm as actual output power. This value of 3 dBm can be increased or decreased in given temperature ranges using the offsets (0.5 dBm/count) in the pu8PowerManagement array." in the description item section Section 3.2.29.

Revision history*...continued*

Rev	Date	Description
		FXTH87xxx1xFWUG v.2.2 modifications (Continued) • Revised the description of Section 3.2.29 as follows: – Removed the item "Set the RF power setting (RFCFR2_PWR) dynamically based on voltage, temperature, and current carrier frequency (when BIT5 of u8PowerManagement is set). The power domains for this function are described in the FXTH87xxxx family of data sheets." – Removed the item "Similar to the case above, the user can specify a target power region with an offset." • Updated Table 24 in Section 3.2.29 as follows: – Revised the index values from 0, 1, 2, 3 and 4 to pu8PowerManagement[0], pu8Power Management[1], pu8PowerManagement[2], pu8PowerManagement[3], and pu8Power Management[3]. – Revised the descriptions for index values 0-4. – Added three new rows with index values/descriptions for pu8PowerManagement[5], pu8Power Management[6] and pu8PowerManagement[7]. • Revised description item of Section 3.2.39 as follows: – Added "Be careful to call the function upon reception of the first data byte (LFDRF flag) and not upon detection of the ID (LFIDF flag) in case the LFIDIE is enabled." after the first sentence of the description. – Added " In order to leave the routine as soon as possible after reception of all the data bytes it is recommended to enable the LF error interrupt (LFERIE). In summary, the two necessary interrupts to be enabled are LFDRIE and LFERIE." at the end of the description.
v.2.1	2014-10	• Changed Bit 4 and Bit 3 values for register LFCTRLD from 0 to 1 in Table 5 in Section 2.6. • Changed Bit 4 and Bit 3 values for register LFCTRLD from 0 to 1 in Table 6 in Section 2.6.
v.1.01	2014-10	• Removed "Xtrinsic" from document title. • Removed sentence "Some functionality may not be present in all derivatives." from the first paragraph of Section 1. • Revised the content of Section 2.1.1 from "This global variable keeps track of interrupts that have occurred. FXTH87xx11 and FXTH87xx12 Embedded Firmware uses it to keep track of expected interrupts. It can also be utilized by the user for its own purposes. The TPMS_INTERRUPT_FLAG is not cleared automatically. Users must clear this variable after power-on-reset. Table 1 shows the TPMS_INTERRUPT_FLAG format. The trigger condition column describes what is necessary for that flag to be set." to "Table 1 summarizes all global variables used by Freescale firmware and their locations. Developers must account for these variables when creating new user firmware." and added Table 2. • Added Section 2.1.2. • Updated references from "G-Cell" and "G-cell" to "g-cell" throughout document. • Added Section 2.6. • Revised Approx. Duration item from "872 usec" to "204 usec" in Section 3.2.7. • Revised second item under Input Parameters adding "In case the acceleration is too high or too low and function has run out of offset steps, a value of 255 ("0 - 1") or 16 ("15 + 1") shall be returned." in Section 3.2.12. • Revised Approx. Duration item from "204 usec" to "955 usec" in Section 3.2.10. • Revised second item under Input Parameters in Section 3.2.12 to read "UINT8* u8Offset: Pointer to initial step to load. Valid offset steps range from 0 - 15 and are described in the device's data sheet. An updated offset value is returned at the end of the function. In case the acceleration is too high or too low and function has run out of offset steps, a value of 255 ("0 - 1") or 16 ("15 + 1") shall be returned." • Revised Approx. Duration item from "204 usec" to "955 usec" in Section 3.2.13.
v.1.0	2014-07	Initial release.

1 Introduction

This document is divided into three sections: This introduction, a section describing global variables and standard formats used throughout the functions, and a third section describing each function.

2 Globals and formats

2.1 Global variables

Table 1 summarizes all global variables used by NXP firmware and their locations. Developers must account for these variables when creating new user firmware.

Table 1. Global variables and their locations

Name	Address	Reference
TPMS_CONT_ACCEL_GLOBAL_VARIABLE	$8E	Section 2.1.2
TPMS_INTERRUPT_FLAG	$8F	Section 2.1.1

2.1.1 TPMS_INTERRUPT_FLAG

This global variable keeps track of interrupts that have occurred. FXTH87xx11 and FXTH87xx12 Embedded Firmware uses it to keep track of expected interrupts. It can also be utilized by the user for its own purposes. If an LFR interrupt occurs while a firmware function is under execution, the LFR User Interrupt Vector will not be accessed, and the bit 2 below will be the only indication available. Users should check this bit, either prior to entering the firmware function or after the firmware function, to assure interrupts are not missed. Also, a number of firmware functions utilize the Stop1 or Stop4 modes, which disable the hardware Watch-dog block. In order to provide a back-up recovery, users should utilize either the RTI or PWU which can be programmed for interrupt if a software or firmware routine has consumed too much time. The Watch-dog is automatically restarted when the program goes back in RUN mode.

The TPMS_INTERRUPT_FLAG is not cleared automatically. Users *must* clear this variable after power-on-reset.

Table 2 shows the TPMS_INTERRUPT_FLAG format. The trigger condition column describes what is necessary for that flag to be set.

Table 2. TPMS_INTERRUPT_FLAG format and trigger conditions

Flag	BIT	Trigger condition
LVD Interrupt	7	LVD interrupt entered.
PWU Interrupt	6	PWU interrupt entered.
TOF Interrupt	5	TOF interrupt entered.
LFR Error Interrupt	4	LFR interrupt entered and LFERF bit of the LFS register is set.
ADC Interrupt	3	ADC interrupt entered.
LFR Interrupt	2	LFR interrupt entered and LFERF bit of the LFS register is clear.
RTI Interrupt	1	RTI interrupt entered.
KBI Interrupt	0	KBI interrupt entered.

TPMS_INTERRUPT_FLAG is 1 byte long and is located at address $8F. Users must account for this variable when developing for the FXTH87xx11 and FXTH87xx12.

2.1.2 TPMS_CONT_ACCEL_GLOBAL_VARIABLE

TPMS_CONT_ACCEL_GLOBAL_VARIABLE is 1 byte long and is located at address $8E. Users must account for this variable when developing for the FXTH87xx11 and FXTH87xx12, and can ignore the contents of said variable as long as it is not overwritten. It is used internally by the TPMS_READ_ACCEL family of functions and its purpose it to communicate the next measurement's sample rate when the *u8Avg* argument is set to a value greater than 2.

2.2 Measurement error format

2.2.1 Definition of signal ranges

Each measured parameter (pressure, voltage, temperature, and acceleration) results from an ADC conversion of an analog signal. This ADC result may then be passed by the firmware to the application software as either the raw ADC result or further compensated and scaled for an output between one and the maximum digital value minus one. The minimum digital value of zero and the maximum digital value are reserved as error codes.

The signal ranges and their significant data points are shown in Figure 1. In this definition, the signal source would normally output a signal between S_{INLO} and S_{INHI}. Due to process, temperture, and voltage variations this signal may increase its range to S_{INMIN} to S_{INMAX}. In all cases the signal will be between the supply rails, so that the ADC will convert it to a range of digital numbers between 0 and 1023 (or 0 and 4095 in the case of temperature readings). These digital numbers will have corresponding D_{INMIN}, D_{INLO}, D_{INHI}, D_{INMAX} values. The ADC digital value is taken by the firmware and compensated and scaled to give the required output code range.

Digital input values below D_{INMIN} and above D_{INMAX} are immediately flagged as being out of range and generate error bits and the output is forced to the corresponding railed-high or railed-low values.

Digital values below D_{INLO} (but above D_{INMIN}) or above D_{INHI} (but not D_{INMAX}) will most likely cause an output that would be less than 1 or greater than 510, respectively. These cases are considered underflow or overflow, respectively. Underflow results will be forced to a value of 1. Overflow results will be forced to a value of 510.

Digital values between D_{INLO} and D_{INHI} will normally produce an output between 1 to 510 (for a 9-bit result). In some isolated cases due to compensation calculations and rounding the result may be less than 1 or greater than 510, in which case the underflow and overflow rule mentioned above is used.

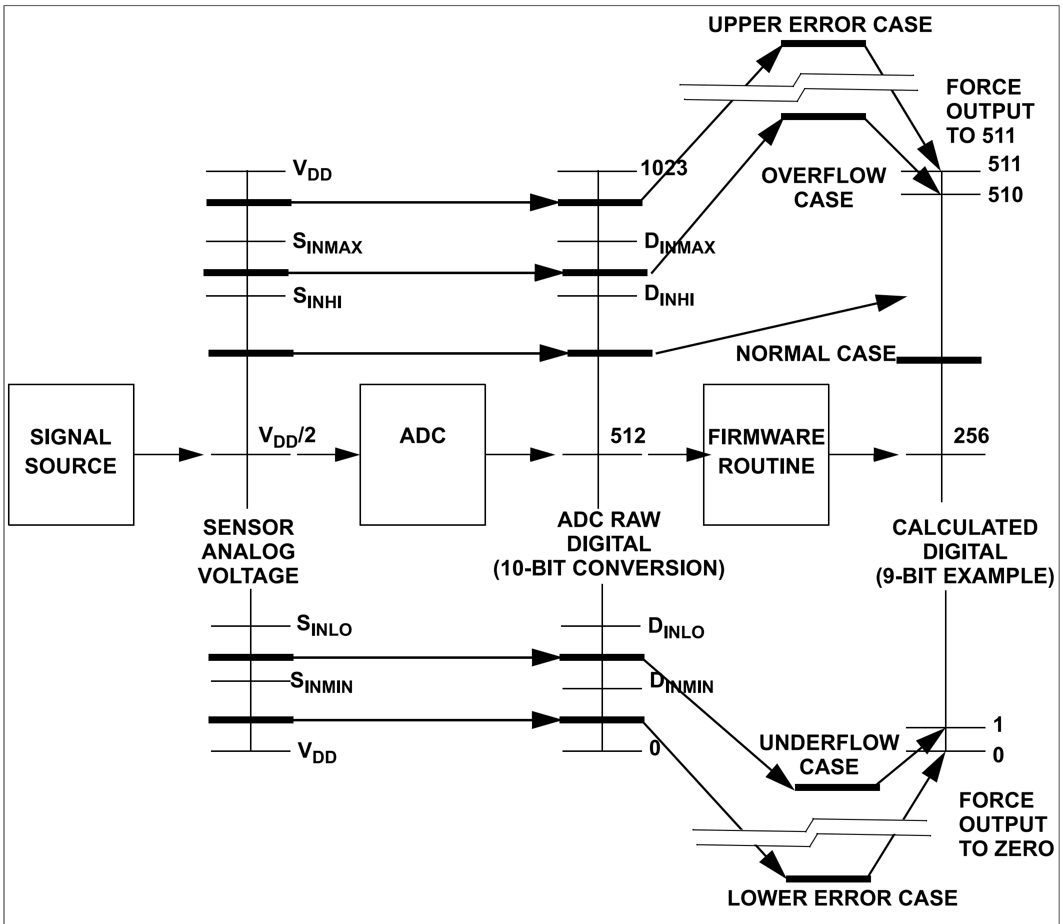

Figure 1. Measurement signal range definitions

2.2.2 Error status format

FXTH87xx11 and FXTH87xx12 Embedded Firmware functions that return a status byte commonly do so using the error fields described in Table 3.

Table 3. Error status fields

Field	Description
BIT7 - ADCERR	ADC Error - This status bit indicates that an error was detected when performing an ADC test within the TPMS_WIRE_AND_ADC_CHECK routine, or when the measurement could not be acquired after the execution of a TPMS_READ function. 0 ADC operating as expected. 1 ADC returned unexpected reading.
BIT6 - TERR	Temperature Measurement Error- This status bit indicates that an error was detected by an ADC reading of the temperature sensor that is outside the normally accepted range. 0 Temperature error not detected in last firmware subroutine call. 1 Temperature error detected in last firmware subroutine call.

Table 3. Error status fields*...continued*

Field	Description
BIT5 - VERR	Voltage Measurement Error- This status bit indicates that an error was detected by an ADC reading of the voltage reference that is outside the normally accepted range. 0 Voltage error not detected in last firmware subroutine call. 1 Voltage error detected in last firmware subroutine call.
BIT4 - AZERR	Z-Axis Accelerometer Measurement Error (if applicable)- This status bit indicates that an error was detected by a bonding wire failure to the g-cell or an ADC reading of the Z-axis accelerometer that is outside the normally accepted range. 0 Acceleration error not detected in last firmware subroutine call. 1 Acceleration error detected in last firmware subroutine call.
BIT3 - AXERR	X-Axis Accelerometer Measurement Error (if applicable)- This status bit indicates an error was detected by a bonding wire failure to the g-cell or an ADC reading of the X-axis accelerometer that is outside the normally accepted range. 0 Acceleration error not detected in last firmware subroutine call. 1 Acceleration error detected in last firmware subroutine call.
BIT2 - PERR	Pressure Measurement Error- This status bit indicates that an error was detected by a parity fault in the P-Chip trim, bonding wire failure to the P-Chip or an ADC reading of the pressure that is outside the normally accepted range. 0 Pressure error not detected in last firmware subroutine call. 1 Pressure error detected in last firmware subroutine call.
BIT1 - BONDERR	Bond Wire Error- This status bit indicates that an error was detected in any of the bond wire checks of the g-cell or P-cell. 0 Bond wire error not detected in last firmware subroutine call. 1 Bond wire error detected in last firmware subroutine call
BIT0 - OVFLOW	Calculation Overflow/Underflow- This status bit indicates that a compensated measurement of pressure, temperature, voltage, or acceleration resulted in a digital output code outside the expected range. The output value will be clipped to the nearest highest or lowest allowed value and the status bit will be set. 0 Overflow/underflow not detected in last firmware subroutine call. 1 Overflow/underflow detected in last firmware subroutine call.

2.3 Universal uncompensated measurement array (UUMA) format

The FXTH87xx11 and FXTH87xx12's measurement routines are divided into two subsets: routines that return uncompensated measurements, and routines that take uncompensated measurements as arguments and return compensated measurements.

In order to be consistent and keep the number of CPU cycles down, all uncompensated measurement routines will return data following the array format described in Table 4, and all compensating routines will take data from the same array.

Table 4. Universal uncompensated measurement array

Index	Content
0	Uncompensated Voltage
1	Uncompensated Temperature

Table 4. Universal uncompensated measurement array*...continued*

Index	Content
2	Uncompensated Pressure
3	Uncompensated X-Axis Acceleration
4	Uncompensated Z-Axis Acceleration

This array is referred to as Universal Uncompensated Measurement Array (UUMA). It can be located anywhere the user decides.

Each element must be 16-bits long (two bytes) regardless of what the actual bit-width of the measurement is.

Each individual uncompensated measurement routine will only update its corresponding item. For example, calling the TPMS_READ_VOLTAGE routine will only modify the voltage element of the array. The rest will remain unchanged.

Compensation routines do not modify any elements in the UUMA.

2.4 Simulated SPI interface signal format

The FXTH87xx11 and FXTH87xx12 include three routines (TPMS_MSG_INIT, TPMS_MSG_READ and TPMS_MSG_WRITE) that, when used together, allow the user to perform serial communication with the device through a simulated SPI interface.

The following assumptions are made:

- Only two pins are used: PTA0 for data (both incoming and outgoing) and PTA1 for clock. No slave select is included by default, but the user may use any other pin if required.
- The data pin has a pull-up resistor enabled.
- The FXTH87xx11 and FXTH87xx12 will be a master device (the FXTH87xx11 and FXTH87xx12 will provide the clock).
- Data can be read/written 8-bits at a time.
- Speed of the interface is dependent on bus clock settings.
- Data is transferred MSB first.
- A single line will be used for both sending and receiving data (BIDIROE = SET according to NXP nomenclature).
  - At the clock's rising edge, the master will place data on the pin. It will be valid until the clock's falling edge. The slave must not drive the line during this period.
  - At the clock's falling edge, the master will make the data pin an input and will "listen" for data. The slave must then place data on the data line until the clock's rising edge.
- Clock Polarity = 0 (Normally low).
- Clock Phase = 1 (First half is high).

Figure 2 shows the details of the simulated SPI interface.

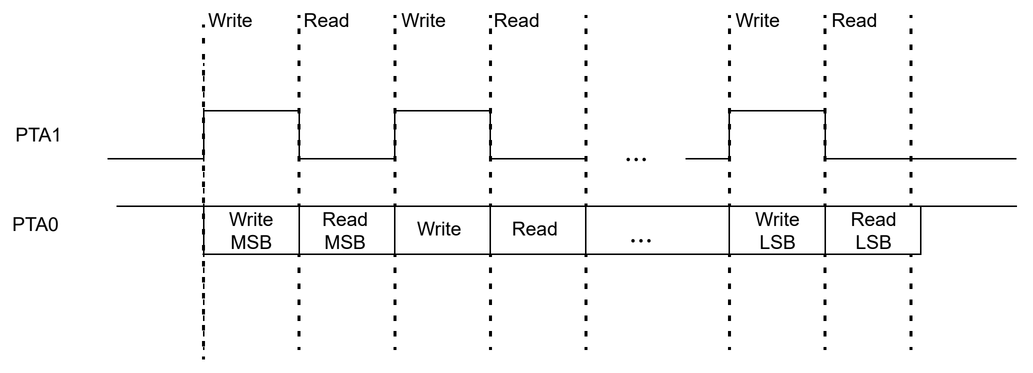

Figure 2. Description of the physical layer on the FXTH87xx11 and FXTH87xx12 simulated SPI interface

For further information on the use of the Simulated SPI interface routines, refer to Section 3.2.30, Section 3.2.31, and Section 3.2.32.

2.5 Rapid decompression event array (T_RDE) format

The FXTH87xx11 and FXTH87xx12 include a routine called TPMS_RDE_ADJUST_PRESSURE that requires a pointer to an array of elements using a custom format called T_RDE. Said format is easily manageable using a typedef instruction as shown in the following example.

Example 1. Sample typedef for a T_RDE array

```
typedef struct
{
UINT16 u16CompPress;      /* I/O 9-bit Compensated pressure reading   */
UINT8  u8ElapsedTime;     /* I Elapsed time from previous reading     */
UINT16 u16WAvg;           /* O Weighed average for running pressure   */
UINT8  u8PRes;            /* O 8-bit pressure reserve value           */
UINT8  u8PMin;            /* O 8-bit minimum pressure value           */
UINT8  u8RDEStatusFlags;  /* O Contains flags for Clock and RDE Event */
UINT16 u16RDEBailTimeOut; /* O Seconds to 60 mins bail-out            */
UINT8  u8RDETimeToAvg;    /* O Seconds to next averaging event        */
} T_RDE;
```

As shown by the comments, only the u16CompPress and u8ElapsedTime elements of this array should be edited by the user; the rest will be updated by the TPMS_RDE_ADJUST_PRESSURE function.

In order for TPMS_RDE_ADJUST_PRESSURE to work correctly, the T_RDE variable must be declared as a global and must reside in an NVM location.

For more information on TPMS_RDE_ADJUST_PRESSURE, refer to Section 3.2.47.

2.6 LFR registers initialized by firmware

Some LFR registers are touched by firmware when taking the reset vector and before giving control to the user. The goal of this action is to configure the LFR module in the best-known configuration for Manchester-encoded reception.

LFR registers will be configured differently depending on the user-selected sensitivity. Table 5 and Table 6 describe these settings.

Table 5. Customer-configurable TPMS7 LF registers with SENS = 1

Page-0	Bit name							
Register name	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LFCTL1	LFEN	SRES	CARMOD	PAGE	IDSEL		SENS	
LFCTL2	LFSTM				LFONTM			
LFCTL3	LFDO	TOGMOD	SYNC		LFCDTM			
LFCTL4	LFDRIE	LFERIE	LFCDIE	LFIDIE	DECEN	VALEN	TIMOUT	
LFS	LFDRF	LFERF	LFCDF	LFIDF	LFOVF	LFEOMF	LPSM	LFIAK
LFDATA	RXDATA							
LFIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
LFIDH	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

Page-1	Bit name							
Register name	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LFCTL1	LFEN	SRES	CARMOD	PAGE	IDSEL		SENS = 1	
LFCTRLE	—	—	—	—	—	0	0	0
LFCTRLD	1	0	DEQS	1	1	1	0	0
LFCTRLC	0	0	0	1	AZEN	LOWQ		DEQEN
LFCTRLB	1	1	LFFAF	LFCAF	LFPOL	1	1	0
LFCTRLA	—	—	—	—	LFCC			
TRIM1	—	—	—	—	—	—	—	—
TRIM2	—	—	—	—	—	—	—	—

▨	Shaded cells show register touched by firmware; loaded value is displayed.

Table 6. Customer-configurable TMPS and LF registers with SENS = 2

Page-0	Bit name							
Register name	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LFCTL1	LFEN	SRES	CARMOD	PAGE	IDSEL		SENS	
LFCTL2	LFSTM				LFONTM			
LFCTL3	LFDO	TOGMOD	SYNC		LFCDTM			
LFCTL4	LFDRIE	LFERIE	LFCDIE	LFIDIE	DECEN	VALEN	TIMOUT	
LFS	LFDRF	LFERF	LFCDF	LFIDF	LFOVF	LFEOMF	LPSM	LFIAK
LFDATA	RXDATA							
LFIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
LFIDH	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

Page-1	Bit name							
Register name	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LFCTL1	LFEN	SRES	CARMOD	PAGE	IDSEL		SENS = 2	
LFCTRLE	—	—	—	—	—	0	0	0
LFCTRLD	1	0	DEQS	1	1	1	0	0

Page-1	Bit name							
Register name	7	6	5	4	3	2	1	0
LFCTRLC	0	0	0	1	AZEN	LOWQ		DEQEN
LFCTRLB	1	1	LFFAF	LFCAF	LFPOL	1	1	0
LFCTRLA	—	—	—	—	LFCC			
TRIM1	—	—	—	—	—	—	—	—
TRIM2	—	—	—	—	—	—	—	—

▨	Shaded cells show register touched by firmware; loaded value is displayed.

3 Firmware functions

3.1 Firmware jump table

The FXTH87xx11 and FXTH87xx12 devices contain an embedded firmware function jump table to allow programmers to reference any function through a function pointer to an absolute address. This helps isolate NXP firmware from the user's application. Table 7 shows a list of all firmware functions and their address.

For a description of how to implement pointers to fixed addresses using the C language, refer to Manual_Compiler_HC08.pdf (part of the CodeWarrior package).

Table 7. FXTH87xx11 and FXTH87xx12's firmware function jump table

Absolute Address	Return type	Function	Reference
$E000	void	TPMS_RESET	Section 3.2.1
$E003	UINT8	TPMS_READ_VOLTAGE	Section 3.2.2
$E006	UINT8	TPMS_COMP_VOLTAGE	Section 3.2.3
$E009	UINT8	TPMS_READ_TEMPERATURE	Section 3.2.4
$E00C	UINT8	TPMS_COMP_TEMPERATURE	Section 3.2.5
$E00F	UINT8	TPMS_READ_PRESSURE	Section 3.2.6
$E012	UINT8	TPMS_COMP_PRESSURE	Section 3.2.7
$E015	UINT8	TPMS_READ_ACCELERATION_X	Section 3.2.8
$E018	UINT8	TPMS_READ_DYNAMIC_ACCEL_X	Section 3.2.9
$E01B	UINT8	TPMS_COMP_ACCELERATION_X	Section 3.2.10
$E01E	UINT8	TPMS_READ_ACCELERATION_Z	Section 3.2.11
$E021	UINT8	TPMS_READ_DYNAMIC_ACCEL_Z	Section 3.2.12
$E024	UINT8	TPMS_COMP_ACCELERATION_Z	Section 3.2.13
$E027	UINT8	TPMS_READ_ACCELERATION_XZ	Section 3.2.14
$E02A	UINT8	TPMS_READ_DYNAMIC_ACCEL_XZ	Section 3.2.15
$E02D	UINT8	TPMS_COMP_ACCELERATION_XZ	Section 3.2.16
$E030	UINT8	TPMS_READ_V0	Section 3.2.17
$E033	UINT8	TPMS_READ_V1	Section 3.2.18
$E036	UINT8	TPMS_LFOCAL	Section 3.2.19

Table 7. FXTH87xx11 and FXTH87xx12's firmware function jump table*...continued*

Absolute Address	Return type	Function	Reference
$E039	UINT8	TPMS_MFOCAL	Section 3.2.20
$E03C	void	TPMS_RF_ENABLE	Section 3.2.21
$E03F	void	TPMS_RF_RESET	Section 3.2.22
$E042	void	TPMS_RF_READ_DATA	Section 3.2.23
$E045	void	TPMS_RF_READ_DATA_REVERSE	Section 3.2.24
$E048	void	TPMS_RF_WRITE_DATA	Section 3.2.25
$E04B	void	TPMS_RF_WRITE_DATA_REVERSE	Section 3.2.26
$E04E	void	TPMS_RF_CONFIG_DATA	Section 3.2.27
$E051	—	Reserved	—
$E054	void	TPMS_RF_SET_TX	Section 3.2.28
$E057	void	TPMS_RF_DYNAMIC_POWER	Section 3.2.29
$E05A	void	TPMS_MSG_INIT	Section 3.2.30
$E05D	UINT8	TPMS_MSG_READ	Section 3.2.31
$E060	UINT8	TPMS_MSG_WRITE	Section 3.2.32
$E063	UINT8	TPMS_CHECKSUM_XOR	Section 3.2.33
$E066	UINT8	TPMS_CRC8	Section 3.2.34
$E069	UINT16	TPMS_CRC16	Section 3.2.35
$E06C	UINT16	TPMS_SQUARE_ROOT	Section 3.2.36
$E06F	void	TPMS_READ_ID	Section 3.2.37
$E072	void	TPMS_LF_ENABLE	Section 3.2.38
$E075	UINT8	TPMS_LF_READ_DATA	Section 3.2.39
$E078	UINT8	TPMS_WIRE_AND_ADC_CHECK	Section 3.2.40
$E07B	void	TPMS_FLASH_WRITE	Section 3.2.41
$E07E	UINT16	TPMS_FLASH_CHECK	Section 3.2.42
$E081	UINT8	TPMS_FLASH_ERASE	Section 3.2.43
$E084	UINT8	TPMS_FLASH_PROTECTION	Section 3.2.44
$E087	—	Reserved	—
$E08A	void	TPMS_MULT_SIGN_INT16	Section 3.2.45
$E08D	UINT16	TPMS_WAVG	Section 3.2.46
$E090	UINT8	TPMS_RDE_ADJUST_PRESSURE	Section 3.2.47

3.2 Function description

The following function descriptions include stack sizes and approximate duration.

Stack sizes have been calculated by executing each routine and measuring the amount of memory utilized. Unless otherwise noted, they represent the maximum stack the function will utilize.

Duration estimates are performed on one part at room temperature. They are intended to serve as a guideline for typical execution time.

3.2.1 void TPMS_RESET(void)

- **Description:** This function is called when taking the reset vector. It will reset the Stack Pointer to the last RAM location and jump to the location stored by the user in $DFFE:DFFF. No further initialization is performed.
- **Stack size:** 3 bytes
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** Stack
- **Input Parameters:**
  - None
- **Returns:**
  - void

3.2.2 UINT8 TPMS_READ_VOLTAGE(UINT16 *u16UUMA)

- **Description:** Performs a 10-bit uncompensated voltage measurement and places it in the UUMA. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit.
  - If the ADC value is over or under the normal operating condition, the "voltage error" status flag will be set. The expected voltage result will be forced to either "0" or "1023." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
  - Measurements below 2.1 V are not guaranteed for accuracy.
- **Stack size:** 22 bytes
- **Approx. Duration:** 102 µs
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** ADC, band gap.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 10-bit uncompensated voltage result will be updated.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 8.

Table 8. Valid output conditions for TPMS_READ_VOLTAGE

u8Status Value	Measurement Value	Condition
$20	$03FF	Uncompensated voltage reading outside the valid range (high).
$20	$0000	Uncompensated voltage reading outside the valid range (low).

Table 8. Valid output conditions for TPMS_READ_VOLTAGE*...continued*

u8Status Value	Measurement Value	Condition
$80	Undefined	Uncompensated voltage reading not acquired.
$00	Between $0001 - $03FE	Valid uncompensated voltage reading.

Warning: *The band gap bit (BIT0 in the SPMSC1 register) must be set prior to calling this function for results to be valid.*

3.2.3 UINT8 TPMS_COMP_VOLTAGE(UINT8 *u8CompVoltage, *UINT16 u16UUMA)

- **Description:** Performs an 8-bit compensated voltage measurement. It is the user's responsibility to ensure that updated and valid uncompensated voltage reading is available in the UUMA for this routine to return a meaningful value.
  - If Vout < 2.1 V, u8Voltage will be 1 and the "over/underflow" status flag will be set.
  - Measurements below 2.1 V are not guaranteed for accuracy.
  - If Vout ≥ 3.7 V, result will be $FE and the "over/underflow" status flag will be set.
  - For repeatability data, refer to the FXTH87xxxx family of data sheets.
- **Stack size:** 31 bytes
- **Approx. Duration:** 204 μs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** UUMA
- **Input Parameters:**
  - UINT8 *u8Voltage: Updated 8-bit compensated voltage result.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated voltage will be utilized from this array.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 9.

Table 9. Valid output conditions for TPMS_COMP_VOLTAGE

u8Status Value	Measurement Value	Condition
$01	$FE	Compensated voltage reading outside the valid range (high).
$01	$01	Compensated voltage reading outside the valid range (low).
$00	Between $01 - $FE	Valid compensated voltage reading.

3.2.4 UINT8 TPMS_READ_TEMPERATURE(UINT16 *u16UUMA)

- **Description:** Performs a 12-bit uncompensated temperature measurement and places it in the UUMA. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit.
  - If the ADC value is over or under the normal operating condition, the "temperature error" status flag will be set. The expected temperature result will be forced to either "0" or "4095." (rail high or rail low). If the LVWF (Low Voltage Warning Flag) hardware bit is set, it will flag it accordingly as well.

  - If the ADC value is over or under the normal operating condition, the "temperature error" status flag will be set. The expected temperature result will be forced to either "0" or "4095." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
- **Stack size:** 17 bytes
- **Approx. Duration:** 219 µs
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** ADC, band gap.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 12-bit uncompensated temperature result will be updated.
- **Returns:** UINT8 u8Status:Valid error flags/outputs are described in Table 10.

.

Table 10. Valid output conditions for TPMS_READ_TEMPERATURE

u8Status Value	Measurement Value	Condition
$40	$0FFF	Uncompensated temperature reading outside the valid range (high).
$40	$0000	Uncompensated temperature reading outside the valid range (low).
$60	$0FFF	Uncompensated temperature reading outside the valid range (high), and LVWF set.
$60	$0000	Uncompensated temperature reading outside the valid range (low), and LVWF set.
$80	Undefined	Uncompensated temperature reading not acquired.
$A0	Undefined	Uncompensated temperature reading not acquired, and LVWF set.
$00	Between $0001 - $0FFE	Valid uncompensated temperature reading.
$20	Between $0001 - $0FFE	Valid uncompensated temperature reading, LVWF set.

Warning: *The band gap bit (BIT0 in the SPMSC1 register) must be set prior to calling this function for results to be valid.*

3.2.5 UINT8 TPMS_COMP_TEMPERATURE(UINT8 *u8Temp, UINT16 *u16UUMA)

- **Description:** Performs an 8-bit compensated temperature measurement. It is the user's responsibility to ensure that updated and valid uncompensated temperature reading is available in the UUMA for this routine to return a meaningful value.
  - If Tout < -40 °C, u8Temp will be 1 and the "over/underflow" status flag will be set.
  - If Tout ≥ 200 °C, u8Temp will be $FE and the "over/underflow" status flag will be set.
- **Stack size:** 30 bytes.
- **Approx. Duration:** 231 µs.

- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** UUMA
- **Input Parameters:**
  - UINT8 *u8Temp: Updated 8-bit compensated temperature result.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated temperature will be utilized from this array.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 11.

Table 11. Valid output conditions for TPMS_COMP_TEMPERATURE

u8Status Value	Measurement Value	Condition
$01	$FE	Compensated temperature reading outside the valid range (high).
$01	$01	Compensated temperature reading outside the valid range (low).
$00	Between $01 - $FE	Valid compensated temperature reading.

3.2.6 UINT8 TPMS_READ_PRESSURE(UINT16 *u16UUMA, UINT8 u8Avg)

- **Description:** Performs a 10-bit uncompensated pressure measurement and places it in the UUMA. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit. If the LVWF (Low Voltage Warning Flag) hardware bit is set, it will flag it accordingly as well.
  - If the ADC value is over or under the normal operating condition, the "pressure error" status flag will be set. The expected pressure result will be forced to either "0" or "1023." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
- **Stack size:** 26 bytes
- **Approx. Duration:** 2872 μs (avg of 1); 10670 μs (avg of 4).
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 10-bit uncompensated pressure result will be updated.
  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, or 16.
- **Returns:** UINT8 u8Status: Valid error flags/outputs are described in Table 12.

Table 12. Valid output conditions for TPMS_READ_PRESSURE

u8Status Value	Measurement Value	Condition
$04	$03FF	Uncompensated pressure reading outside the valid range (high).
$04	$0000	Uncompensated pressure reading outside the valid range (low).
$24	$03FF	Uncompensated pressure reading outside the valid range (high), and LVWF set.
$24	$0000	Uncompensated pressure reading outside the valid range (low), and LVWF set.
$80	$0000	Uncompensated pressure reading not acquired
$A0	$0000	Uncompensated pressure reading not acquired, and LVWF set.
$00	Between $0001 - $03FE	Valid uncompensated pressure reading.
$20	Between $0001 - $03FF	Valid uncompensated pressure reading, and LVWF set.

3.2.7 UINT8 TPMS_COMP_PRESSURE(UINT16 *u16CompPressure, UINT16 *u16UUMA)

- **Description:** Performs a 9-bit compensated pressure measurement. It is the user's responsibility to ensure that updated and valid uncompensated voltage, temperture, and pressure readings are available in the UUMA for this routine to return a meaningful value.
  - If either the temperature or supply voltage measurements inherent to this function result in a fault, the pressure reading will be forced to 0 and the appropriate pressure, temperture, and/or voltage flags will be set in the status flag.
  - If Pout < 100 kPa, the "over/underflow" status flag will be set, and u16CompPressure will be forced to $001.
  - If Pout ≥ 900 kPa, u16CompPressure will be $1FE and the "over/underflow" status flag will be set.
  - If the passed uncompensated voltage measurement is estimated to be under the guaranteed operational region, the routine will set the "Voltage" status flag. The accuracy of the returned value is not guaranteed.
  - For repeatability data, refer to the FXTH87xxxx family of data sheets.
- **Stack size:** 46 bytes
- **Approx. Duration:** 872 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** UUMA.
- **Input Parameters:**
  - UINT16 *u16Pressure: Updated 9-bit compensated pressure result.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated voltage, temperture, and pressure will be taken from this array.
- **Returns:** UINT8 u8Status: Valid error flags/outputs are described in Table 13.

Table 13. Valid output conditions for TPMS_COMP_PRESSURE

u8Status Value	Measurement Value	Condition
$01	$01FE	Compensated pressure reading outside the valid range (high).
$01	$0001	Compensated pressure reading outside the valid range (low).
$21	$01FE	Compensated pressure reading outside the valid range (high), and uncompensated voltage suspected to be below valid operating range for this function.
$21	$0001	Compensated pressure reading outside the valid range (low), and uncompensated voltage suspected to be under below operating range for this function.
$20	Between $0001 - $01FE	Uncompensated voltage suspected to be below valid operating range for this function; The compensated reading is not guaranteed for accuracy.
$00	Between $0001 - $01FE	Valid compensated pressure reading.

3.2.8 UINT8 TPMS_READ_ACCELERATION_X(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffset)

- **Description:** Performs an uncompensated 10-bit measurement. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit. If the LVWF (Low Voltage Warning Flag) hardware bit is set, it will flag it accordingly as well.
  - If the ADC value is over or under the normal operating condition, the "acceleration error" status flag will be set. The expected acceleration result will be forced to either "0" or "1023." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
- **Stack size:** 31 bytes
- **Approx. Duration:** 2901 μs (500 Hz filter, 1 reading), 4255 μs (250 Hz filter, 1 reading).
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 10-bit uncompensated acceleration result will be updated.
  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, or 16.
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8 u8DynamicOffset: Selects the offset setting for the appropriate acceleration reading. Default is 6.

.

Table 14. u8DynamicOffset valid values

Offset Index	Offset when result of function is 256 counts assuming stand trim. Span does not change.
0	-70 g
1	-60 g
2	-50 g
3	-40 g
4	-30 g
5	-20 g
6	-10 g
7	0 g (default)
8	10 g
9	20 g
10	30 g
11	40 g
12	50 g
13	60 g
14	70 g
15	80 g

- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 15.

Table 15. Valid output conditions for TPMS_READ_ACCELERATION_X

u8Status Value	Measurement Value	Condition
$08	$03FF	Uncompensated acceleration reading outside the valid range (high).
$08	$0000	Uncompensated acceleration reading outside the valid range (low).
$28	$03FF	Uncompensated acceleration reading outside the valid range (high), and LVWF set.
$28	$0000	Uncompensated acceleration reading outside the valid range (low), and LVWF set.
$80	$0000	Uncompensated acceleration reading not acquired
$A0	$0000	Uncompensated acceleration reading not acquired, and LVWF set.
$00	Between $0001 - $03FE	Valid uncompensated acceleration reading.
$20	Between $0001 - $03FE	Valid uncompensated acceleration reading, but LVWF set

3.2.9 UINT8 TPMS_READ_DYNAMIC_ACCEL_X(UINT8 u8Filter, UINT8* u8Offset, UINT16* u16UUMA)

- **Description:** This function automatically executes a TPMS_READ_ACCELERATION_X measurement with a given initial dynamic offset. If the result is too high or too low, it will change the dynamic offset value and re-execute TPMS_READ_ACCELERATION_X until a) the result is valid or b) the result is railed high or low and there are no more offset steps. Offset and uncompensated acceleration inside the UUMA are updated.
- **Stack size:** 48 bytes
- **Approx. Duration:** 29065 µs from one extreme to the center; 2816 µs best case.
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8* u8Offset: Pointer to initial offset level to load into SMI according to Table 14. An updated offset value is returned at the end of the function. In case the acceleration is too high or too low and function has run out of offset steps, a value of 255 ("0 - 1") or 16 ("15 + 1") shall be returned.
  - UINT16* Pointer to the Universal Uncompensated Measurement Array. Uncompensated acceleration will be updated accordingly
- **Returns:**
  - UINT8 u8Status: Refer to TPMS_READ_ACCELERATION_X for more information on the format of this status byte.

3.2.10 UINT8 TPMS_COMP_ACCELERATION_X(UINT16 *u16CompAccelX, UINT16* u16UUMA)

- **Description:** Performs a 9-bit compensated acceleration measurement. It is the user's responsibility to ensure that updated and valid uncompensated voltage, temperture, and acceleration readings are available in the UUMA for this routine to return a meaningful value.
  - If u16CompAccel rails low, u16CompAccel will be forced to 1 and the "over/underflow" status flag will be set.
  - If u16CompAccel rails high, u16CompAccel will be forced to $1FE and the "over/underflow" status flag will be set.
  - If the passed uncompensated voltage measurement is estimated to be under the guaranteed operational region, the routine will set the "Voltage" status flag. The accuracy of the returned value is not guaranteed.
  - For repeatability data, refer to the FXTH87xxxx family of data sheets.
- **Stack size:** 55 bytes
- **Approx. Duration:** 955 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.

- **Resources:** UUMA.
- **Input Parameters:**
  - UINT16 *u16AccelX: Updated 9-bit compensated acceleration.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated voltage, temperture, and X-axis acceleration will be taken from this array.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 16.

Table 16. Valid output conditions for TPMS_COMP_ACCELERATION_X

u8Status Value	Measurement Value	Condition
$01	$01FE	Compensated acceleration reading outside the valid range (high).
$01	$0001	Compensated acceleration reading outside the valid range (low).
$21	$01FE	Compensated pressure reading outside the valid range (high), and uncompensated voltage suspected to be below valid operating range for this function.
$21	$0001	Compensated pressure reading outside the valid range (low), and uncompensated voltage suspected to be under below operating range for this function.
$20	Between $0001 - $01FE	Uncompensated voltage suspected to be below valid operating range for this function; The compensated reading is not guaranteed for accuracy.
$00	Between $0001 - $01FE	Valid compensated acceleration reading.

3.2.11 UINT8 TPMS_READ_ACCELERATION_Z(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffset)

- **Description:** Performs an uncompensated 10-bit measurement. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit. If the LVWF (Low Voltage Warning Flag) hardware bit is set, it will flag it accordingly as well.
  - If the ADC value is over or under the normal operating condition, the "acceleration error" status flag will be set. The expected acceleration result will be forced to either "0" or "1023." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
- **Stack size:** 31 bytes
- **Approx. Duration:** 2901 μs (500 Hz filter, 1 reading), 4255 μs (250 Hz filter, 1 reading).
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.

- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 10-bit uncompensated acceleration result will be updated.
  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, or 16.
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8 u8DynamicOffset: Selects the offset setting for the appropriate acceleration reading. Valid range is 0 - 15. Refer to product specification for acceleration ranges for a given offset step.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 17.

Table 17. Valid output conditions for TPMS_READ_ACCELERATION_Z

u8Status Value	Measurement Value	Condition
$10	$03FF	Uncompensated acceleration reading outside the valid range (high).
$10	$0000	Uncompensated acceleration reading outside the valid range (low).
$30	$03FF	Uncompensated acceleration reading outside the valid range (high), and LVWF set.
$30	$0000	Uncompensated acceleration reading outside the valid range (low), and LVWF set.
$80	$0000	Uncompensated acceleration reading not acquired.
$A0	$0000	Uncompensated acceleration reading not acquired, and LVWF set.
$00	Between $0001 - $03FE	Valid uncompensated acceleration reading.
$20	Between $0001 - $03FE	Valid uncompensated acceleration reading, but LVWF set.

3.2.12 UINT8 TPMS_READ_DYNAMIC_ACCEL_Z(UINT8 u8Filter, UINT8* u8Offset, UINT16* u16UUMA)

- **Description:** This function automatically executes a TPMS_READ_ACCELERATION_Z measurement with a given initial dynamic offset. If the result is too high or too low, it will change the dynamic offset value and re-execute TPMS_READ_ACCELERATION_Z until a) the result is valid or b) the result is railed high or low and there are no more offset steps. Offset and uncompensated acceleration inside the UUMA are updated.
- **Stack size:** 48 bytes
- **Approx. Duration:** 29065 µs from one extreme to the center; 2816 µs best case.
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.

- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8* u8Offset: Pointer to initial step to load. Valid offset steps range from 0 - 15 and are described in the device's data sheet. An updated offset value is returned at the end of the function. In case the acceleration is too high or too low and function has run out of offset steps, a value of 255 ("0 - 1") or 16 ("15 + 1") shall be returned.
  - UINT16* Pointer to the Universal Uncompensated Measurement Array. Uncompensated acceleration will be updated accordingly.
- **Returns:**
  - UINT8 u8Status: Refer to TPMS_READ_ACCELERATION_Z for more information on the format of this status byte.

3.2.13 UINT8 TPMS_COMP_ACCELERATION_Z(UINT16 *u16CompAccel, UINT16* u16UUMA)

- **Description:** Performs a 9-bit compensated acceleration measurement. It is the user's responsibility to ensure that updated and valid uncompensated voltage, temperture, and acceleration readings are available in the UUMA for this routine to return a meaningful value.
  - If u16CompAccel rails low, u16CompAccel will be forced to 1 and the "over/underflow" status flag will be set.
  - If u16CompAccel rails high, u16CompAccel will be forced to $1FE and the "over/underflow" status flag will be set.
  - If the passed uncompensated voltage measurement is estimated to be under the guaranteed operational region, the routine will set the "Voltage" status flag. The accuracy of the returned value is not guaranteed.
  - For repeatability data, refer to theFXTH87xxxx family of data sheets.
- **Stack size:** 55 bytes
- **Approx. Duration:** 955 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** UUMA.
- **Input Parameters:**
  - UINT16 *u16Accel: Updated 9-bit compensated acceleration.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated voltage, temperture, and acceleration will be taken from this array.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 18.

Table 18. Valid output conditions for TPMS_COMP_ACCELERATION_Z

u8Status Value	Measurement Value	Condition
$01	$01FE	Compensated acceleration reading outside the valid range (high).
$01	$0001	Compensated acceleration reading outside the valid range (low).

Table 18. Valid output conditions for TPMS_COMP_ACCELERATION_Z*...continued*

u8Status Value	Measurement Value	Condition
$21	$01FE	Compensated pressure reading outside the valid range (high), and uncompensated voltage suspected to be below valid operating range for this function.
$21	$0001	Compensated pressure reading outside the valid range (low), and uncompensated voltage suspected to be under below operating range for this function.
$20	Between $0001 - $01FE	Uncompensated voltage suspected to be below valid operating range for this function; The compensated reading is not guaranteed for accuracy.
$00	Between $0001 - $01FE	Valid compensated acceleration reading.

3.2.14 UINT8 TPMS_READ_ACCELERATION_XZ(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffsetX, UINT8 u8DynamicOffsetZ)

- **Description:** Performs an uncompensated 10-bit measurement. While waiting for the ADC to converge, this function goes into STOP4. If the ADC, for an unexpected reason, fails to converge, this function has a built-in time-out: After five continuous non-ADC interrupts, the function will assume a failed ADC reading, flag it accordingly, and exit.
  - If the ADC value is over or under the normal operating condition, the "acceleration error" status flag will be set. The expected acceleration result will be forced to either "0" or "1023." (rail high or rail low).
  - If the ADC times out with no result, the "ADC error" status flag will be set.
- **Stack size:** 34 bytes
- **Approx. Duration:** 5744 µs (500 Hz filter, 1 reading), 8477 µs (250 Hz filter, 1 reading).
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Only the 10-bit uncompensated acceleration result will be updated.
  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, 16, or 32.
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8 u8DynamicOffsetX: Selects the offset setting for the appropriate acceleration reading. Refer to Table 14 for more information.

  - UINT8 u8DynamicOffsetZ: Selects the offset setting for the appropriate acceleration reading. Refer to Table 17 for more information.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 19.

Table 19. Valid output conditions for TPMS_READ_ACCELERATION_XZ

u8Status Value	Measurement Value	Condition
$08	$03FF	Uncompensated X acceleration reading outside the valid range (high).
$08	$0000	Uncompensated X acceleration reading outside the valid range (low).
$28	$03FF	Uncompensated X acceleration reading outside the valid range (high), and LVWF set.
$28	$0000	Uncompensated X acceleration reading outside the valid range (low) and LVWF set.
$10	$03FF	Uncompensated Z acceleration reading outside the valid range (high).
$10	$0000	Uncompensated Z acceleration reading outside the valid range (low).
$30	$03FF	Uncompensated Z acceleration reading outside the valid range (high), and LVWF set.
$30	$0000	Uncompensated Z acceleration reading outside the valid range (low), and LVWF set.
$18	$03FF	Uncompensated X and Z acceleration readings outside the valid range (high).
$18	$0000	Uncompensated X and Z acceleration readings outside the valid range (low).
$38	$03FF	Uncompensated X and Z acceleration readings outside the valid range (high), and LVWF set.
$38	$0000	Uncompensated X and Z acceleration readings outside the valid range (low), and LVWF set.
$80	Undefined	Uncompensated acceleration reading not acquired.
$A0	Undefined	Uncompensated acceleration reading not acquired, and LVWF set.
$00	Between $0001 - $03FE	Valid uncompensated acceleration reading.
$20	Between $0001 - $03FE	Valid uncompensated acceleration reading, but LVWF set.

3.2.15 UINT8 TPMS_READ_DYNAMIC_ACCEL_XZ(UINT8 u8Filter, UINT8* u8OffsetX, UINT8* u8OffsetZ, UINT16* u16UUMA)

- **Description:** This function automatically executes a TPMS_READ_DYNAMIC_ACCEL_X measurement, followed by a TPMS_READ_DYNAMIC_ACCEL_Z measurement with given initial dynamic offsets. Refer to the description of these functions for more information.
- **Stack size:** 60 bytes

- **Approx. Duration:** 50240 µs from one extreme to the center; 8200 µs best case.
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** SMI, ADC, internal bond wires.
- **Input Parameters:**
  - UINT8 u8FiltSelect: If non-zero, 250 Hz filter enabled. Otherwise, 500 Hz filter selected.
  - UINT8* u8OffsetX: Pointer to initial offset level to load into SMI according to Table 14. An updated offset value is returned at the end of the function.
  - UINT8* u8OffsetZ: Pointer to initial offset level to load into SMI according to Table 17. An updated offset value is returned at the end of the function.
  - UINT16* Pointer to the Universal Uncompensated Measurement Array. Uncompensated acceleration for both axes will be updated accordingly.
- **Returns:**
  - UINT8 u8Status: Refer to TPMS_READ_ACCELERATION_Z for more information on the format of this status byte.

3.2.16 UINT8 TPMS_COMP_ACCELERATION_XZ(UINT16 *u16CompAccel, UINT16* u16UUMA)

- **Description:** This function internally calls TPMS_COMP_ACCELERATION_X followed by TPMS_COMP_ACCELERATION_Z. It places the compensated results in a 2-word array. For more information, refer to the functions afore mentioned.
- **Stack size:** 60 bytes
- **Approx. Duration:** 1910 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await interrupts. It is not affected by interrupts either.
- **Resources:** UUMA
- **Input Parameters:**
  - UINT16 *u16CompAccel: Updated 9-bit compensated X-axis acceleration followed by Z-axis acceleration.
  - UINT16 *u16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). Uncompensated voltage, temperture, and acceleration will be taken from this array.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 20.

Table 20. Valid output conditions for TPMS_COMP_ACCELERATION_XZ

u8Status Value	Measurement Value	Condition
$01	$01FE	Compensated acceleration reading outside the valid range (high).
$01	$0001	Compensated acceleration reading outside the valid range (low).
$21	$01FE	Compensated pressure reading outside the valid range (high), and uncompensated voltage suspected to be below valid operating range for this function.

Table 20. Valid output conditions for TPMS_COMP_ACCELERATION_XZ*...continued*

u8Status Value	Measurement Value	Condition
$21	$0001	Compensated pressure reading outside the valid range (low), and uncompensated voltage suspected to be under below operating range for this function.
$20	Between $0001 - $01FE	Uncompensated voltage suspected to be below valid operating range for this function; The compensated reading is not guaranteed for accuracy.
$00	Between $0001 - $01FE	Valid compensated acceleration reading.

3.2.17 UINT8 TPMS_READ_V0(UINT16 *u16Result, UINT8 u8Avg)

- **Description:** Performs a 10-bit uncompensated measurement at pin PTA0.
- **Stack size:** 23 bytes
- **Approx. Duration:** 107 µs
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** ADC, PTA0.
- **Input Parameters:**
  - UINT16 *u16Result: Updated 10-bit uncompensated measurement.
  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, or 16.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 21.

Table 21. Valid output conditions for TPMS_READ_V0 and TPMS_READ_V1

u8Status Value	Measurement Value	Condition
$01	$0000	Reading not acquired.
$00	Between $0000 - $03FE	Valid reading.

3.2.18 UINT8 TPMS_READ_V1(UINT16 *u16Result, UINT8 u8Avg)

- **Description:** Performs a 10-bit uncompensated measurement at pin PTA1.
- **Stack size:** 23 bytes
- **Approx. Duration:** 107 µs
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** ADC, PTA1.
- **Input Parameters:**
  - UINT16 *u16Result: Updated 10-bit uncompensated measurement.

  - UINT8 u8Avg: Number of measurements to average into one result. The value can be set to 1, 2, 4, 8, or 16.
- **Returns:**
  - UINT8 u8Status: Valid error flags/outputs are described in Table 21.

3.2.19 UINT8 TPMS_LFOCAL(void)

- **Description:** Performs PWU clock calibration. The wake up and periodic reset time can be calibrated more accurately by using the TPMS_LFOCAL firmware subroutine. This subroutine turns on the RFM crystal oscillator and feeds a 500 Hz clock via the DX signal to the TPM1 for one cycle of the LFO. The measured time is used to calculate the correct value for the WDIV0:5 bits for a WCLK period of 1 second. The resulting value for use in the WDIV0:5 bits is returned in the accumulator. The user can decide whether to load the value to the WDIV0:5 bits or store for future reference. The TPMS_LFOCAL subroutine cannot be used while the RFM is transmitting or the TPM1 is being used for another task. This routine will also consume more power due to the crystal oscillator running. This function accesses and writes data to the SIMOPT2 register. Since some of the bits in this register are write-once-only, it should be configured prior to calling this routine.
- **Stack size:** 9 bytes
- **Approx. Duration:** 1747 µs
- **Power Management:** This function executes entirely in RUN mode. It requires the MCU to be configured for 4 MHz bus clock, and the RFM to be enabled but not transmitting prior to making the call.
- **Interrupt Management:** This function does not await any interrupts. It WILL be affected by interrupts.
- **Resources:** TPM, SIMOPT2, RFM
- **Input Parameters:**
  - None
- **Returns:**
  - UINT8 u8WDIV: WDIV compensated value, or $80 if the XTAL was not found.

Warning: *This routine writes to SIMOPT2. Any configuration involving this register must be performed before calling this routine. Prior to calling this routine, the RFM must be enabled. The execution of this routine will change the contents of RFM registers. Specifically note that RF Direct Mode will be selected after its execution.*

3.2.20 UINT8 TPMS_MFOCAL(void)

- **Description:** Performs MFO cross-check verification. This function will measure the bus clock relative to Dx, but first executes a test to verify the presence of the external XTAL. When error is zero, it returns "128." Any deviation from this value should be considered an error. This result can then be used to estimate the error in the RFBT setting. The TPMS_MFOCAL subroutine cannot be used while the RFM is transmitting or the TPM1 is being used for another task. This function accesses and writes data to the SIMOPT2 register. Since some of the bits in this register are write-once-only, it should be configured prior to calling this routine.
- **Stack size:** 9 bytes
- **Approx. Duration:** 1825 µs
- **Power Management:** This function executes entirely in RUN mode. It requires the MCU to be configured for 4 MHz bus clock, and the RFM to be enabled but not transmitting prior to making the call.

- **Interrupt Management:** This function does not await any interrupts. It WILL be affected by interrupts.
- **Resources:** TPM, SIMOPT2, RFM
- **Input Parameters:**
  - None
- **Returns:**
  - UINT8 u8Error: 128 when no error is found. Each LSB away from this value is equal to a 0.78 % error. For example, if u8Error = 125, MFO has a -2.34 % error, or is running at 122 kHz. 255 is reserved as an error code for when the external XTAL is not present.

Warning: *This routine writes to SIMOPT2. Any configuration involving this register must be performed before calling this routine. Prior to calling this routine, the RFM must be enabled. The execution of this routine will change the contents of RFM registers. Specifically note that RF Direct Mode will be selected after its execution.*

3.2.21 void TPMS_RF_ENABLE(UINT8 u8Switch)

- **Description:** This function enables or disables the RF module in the FXTH87xx11 and FXTH87xx12 and transfers adequate PLL trim data to the module. It should be called prior to any other RF operation.
- **Stack size:** 4 bytes
- **Approx. Duration:** 378 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will be affected by interrupts.
- **Resources:** SIMOPT1, RFM
- **Input Parameters:**
  - UINT8 u8Switch: Enable (non-zero) or disable (zero) RFM.
- **Returns:**
  - void.

Warning: *This routine writes to SIMOPT1. Any configuration involving this register must be performed before calling this routine.*

3.2.22 void TPMS_RF_RESET(void)

- **Description:** This function sends a master reset to the RFM and reloads PLL trim values into the module. It requires the RFM to have been enabled previously.
- **Stack size:** 3 bytes
- **Approx. Duration:** 228 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - None
- **Returns:**
  - void

3.2.23 void TPMS_RF_READ_DATA(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer)

- **Description:** This function reads several consecutive bytes from the dedicated RFM buffer registers and copies them to a given address in RAM. It assumes that BUFF0 is location "0". The data is transferred from the LSB bit of the RFM data registers to the LSB of the target memory address (standard data bit order). This function manages the RFM's buffer paged memory.
  - In case the required buffer address is out of bounds, the routine will return "0" for that location.
- **Stack size:** 9 bytes
- **Approx. Duration:** 194 µs (for 8 bytes, switching pages included).
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - UINT8 u8Size: Number of bytes to read.
  - UINT8 *u8RamBuffer: Target memory location.
  - UINT8 u8RFMBuffer: Buffer register (0 to 31) to read.
- **Returns:**
  - void

3.2.24 void TPMS_RF_READ_DATA_REVERSE(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer)

- **Description:** This function reads several consecutive bytes from the dedicated RFM buffer registers and copies them to a given address in RAM. It assumes that BUFF0 is location "0". The data is transferred from the LSB bit of each byte of the RFM data registers to the MSB of each of the bytes of the target memory address (reversed data bit order). This function manages the RFM's buffer paged memory.
  - In case the required buffer address is out of bounds, the routine will return "0" for that location.
- **Stack size:** 10 bytes
- **Approx. Duration:** 236 µs (for 8 bytes, switching pages included).
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - UINT8 u8Size: Number of bytes to read.
  - UINT8 *u8RamBuffer: Target memory location.
  - UINT8 u8RFMBuffer: Buffer register (0 to 31) to read.
- **Returns:**
  - void

3.2.25 void TPMS_RF_WRITE_DATA(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer)

- **Description:** This function copies several consecutive bytes from RAM into the dedicated RFM Output Buffer. It assumes that BUFF0 is location "0". The data is transferred from the LSB bit of RAM to the LSB of the RFM data register (standard data bit order). This function manages the RFM's buffer paged-memory.
  - In case the destination buffer address is out of bounds, the register value will not be written.
- **Stack size:** 8 bytes
- **Approx. Duration:** 182 µs (for 8 bytes, switching pages included).
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - UINT8 u8Size: Number of bytes to write.
  - UINT8 *u8RamBuffer: Source memory location.
  - UINT8 u8RFMBuffer: Starting buffer register (0 to 31) to write.
- **Returns:**
  - void

3.2.26 void TPMS_RF_WRITE_DATA_REVERSE(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer)

- **Description:** This function copies several consecutive bytes from RAM into the dedicated RFM Output Buffer. It assumes that BUFF0 is location "0". The data is transferred from the LSB bit of each byte in RAM to the MSB of each byte in the RFM data register (reversed data bit order). This function manages the RFM's buffer paged-memory.
  - In case the destination buffer address is out of bounds, the register value will not be written.
- **Stack size:** 9 bytes
- **Approx. Duration:** 242 µs (for 8 bytes, switching pages included).
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - UINT8 u8Size: Number of bytes to write.
  - UINT8 *u8RamBuffer: Source memory location.
  - UINT8 8uRFMBuffer: Starting buffer register (0 to 31) to write.
- **Returns:**
  - void

3.2.27 void TPMS_RF_CONFIG_DATA(UINT16 *u16RFParam)

- **Description:** This function is included for backward compatibility with the MPXY8300. This function configures the RFM for transmission. It does not configure inter-frame wait times, which must be configured manually.

- **Stack size:** 4 bytes
- **Approx. Duration:** 32 µs.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  – UINT16* u16RFParam Format as described in Table 22.

Table 22. u16RFParam array format

Index	Description
0	Refer to Table 23 for description.
1	PLLA value.
2	PLLB value

Table 23. Description of element 0 in the u16RFParam array

Bits	Description
15:8	Prescaler value. Described in Data Sheet as RFCR0.
7	End Of Message- If '1', EOM is set, if '0', it is not set.
6	Polarity Bit - If '1', polarity is inverted, If '0', it is non-inverted.
5:4	Not used.
2:3	Encoding value - Refer to the product spec for more information.
1	Frequency selection - If '1', RFM is configured for 434 MHz, if '0', it is configured for 315 MHz.
0	Modulation - If '1', RFM is configured for FSK, if '0'it is configured for OOK.

- **Returns:**
  – void

3.2.28 void TPMS_RF_SET_TX(UINT8 u8BufferSize)

- **Description:** This function allows the RFM to transmit data previously loaded in the buffer. It should be called after the RF module has been enabled and configured.
- **Stack size:** 3 bytes
- **Approx. Duration:** 12 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  – UINT8 u8BufferSize: Number of bits in the buffer -1 (for example, to transmit one bit, u8BufferSize should equal 0).
- **Returns:**
  – void

3.2.29 void TPMS_RF_DYNAMIC_POWER(UINT8 u8CompT, UINT8 u8CompV, UINT8* pu8PowerManagement)

- **Description:** Depending on the passed parameters, this function can:
  - Force the RF power setting (RFCFR2_PWR) to a passed value (when BIT5 of u8PowerManagement is clear).
  - When BIT5 of u8PowerManagement is set, find the best RF power setting (RFCFR2_PWR) dynamically based on voltage, temperature, and current carrier frequency in order to target 3 dBm as actual output power. This value of 3 dBm can be increased or decreased in given temperature ranges using the offsets (0.5 dBm/ count) in the pu8PowerManagement array.
- **Stack size:** 21 bytes
- **Approx. Duration:** 140 µs when calculating dynamic power; 20 µs when power setting is passed.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** RFM
- **Input Parameters:**
  - UINT8 u8CompT: Compensated temperature reading
  - UINT8 u8CompV: Compensated voltage reading
  - UINT8* pu8PowerManagement: This is a pointer to an array of five elements as described below.

Table 24. *pu8PowerManagement format

Index Value	Description
pu8PowerManagement[0]	Dynamic Compensation switch as described in Table 25.
pu8PowerManagement[1]	Offset step for temperatures ≥ 92 °C. Negative values admitted.
pu8PowerManagement[2]	Offset step for 60 °C ≤ temp < 92 °C. Negative values admitted.
pu8PowerManagement[3]	Offset step for 43 °C ≤ temp < 60 °C. Negative values admitted.
pu8PowerManagement[4]	Offset step for 25 °C ≤ temp < 43 °C. Negative values admitted.
pu8PowerManagement[5]	Offset step for 1 °C ≤ temp < 25 °C. Negative values admitted.
pu8PowerManagement[6]	Offset step for –20 °C ≤ temp < 1 °C. Negative values admitted.
pu8PowerManagement[7]	Offset step for –40 °C ≤ temp < –20 °C. Negative values admitted.

Table 25. pu8PowerManagement[0] format

BIT	Description
MSB	Not used.
BIT6	Not used.
BIT5	Dynamic compensation enable. If set, the function will decide what the optimal power setting is based on voltage and temperature; In this case, values stored in the array will be added to the found target. If clear, BIT4:0 will be used to set the power level directly.

Table 25. pu8PowerManagement[0] format*...continued*

BIT	Description
BIT4:0	When BIT5 is clear, the value passed here will be used to set the RF power step directly.

- **Returns:**
  - void

Warning: *The RF Module must be turned on prior to calling this routine.*

3.2.30 void TPMS_MSG_INIT(void)

- **Description:** This function is to be called before using any MSG routine. It initializes PTA1 and PTA0 to their correct initial state for a simulated SPI.
- **Stack size:** 2 bytes
- **Approx. Duration:** 4 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** Pins PTA1 and PTA0.
- **Input Parameters:**
  - None
- **Returns:**
  - void

3.2.31 UINT8 TPMS_MSG_READ(void)

- **Description:** This function is in charge to read any incoming message at a network level via an emulated serial interface on PTA1 and PTA0. As the master, the FXTH87xx11 and FXTH87xx12 manage the clock on PTA1. On falling edge of the clock, the module reads a new data bit on PTA0 (programmed as input), MSB first.
- **Stack size:** 2 bytes
- **Approx. Duration:** 80 µs.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** Pins PTA1 and PTA0.
- **Input Parameters:**
  - None
- **Returns:**
  - UINT8 u8ReadByte: Incoming byte from the emulated serial interface.

3.2.32 UINT8 TPMS_MSG_WRITE(UINT8 u8SendByte)

- **Description:** This function is in charge to write a message at a network level via an emulated serial interface on PTA1 and PTA0. As the master, the FXTH87xx11 and FXTH87xx12 manage the clock on PTA1. On rising edge of the clock, the module puts down a new data bit on PTA0 (programmed as output), MSB first.
- **Stack size:** 2 bytes
- **Approx. Duration:** 80 µs

- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** Pins PTA1 and PTA0.
- **Input Parameters:**
  - UINT8 u8SendByte: Byte to be outputted through the emulated serial interface.
- **Returns:**
  - UINT8 u8ReadByte: Incoming byte from the emulated serial interface.

3.2.33 UINT8 TPMS_CHECKSUM_XOR(UINT8 *u8Buffer, UINT8 u8Size, UINT8 u8Checksum)

- **Description:** Calculates a checksum for the given buffer based on XOR operations.
- **Stack size:** 5 bytes
- **Approx. Duration:** 80 µs for 8 bytes of data.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - UINT8 *u8Buffer: Buffer where data is located.
  - UINT8 u8Size: Size of buffer (in bytes).
  - UINT8 u8Checksum: Previous checksum. This argument is useful when the function is used recursively. It must equal "0" if there is no previous data.
- **Returns:**
  - UINT8 u8NewChecksum: New calculated checksum.

3.2.34 UINT8 TPMS_CRC8(UINT8 *u8Buffer, UINT16 u16SizeInBytes, UINT8 u8Remainder)

- **Description:** Calculates a CRC8 on a portion of the designated area using polynomial x^8 + x^5 + x^3 + x^2 + x + 1.
- **Stack size:** 9 bytes
- **Approx. Duration:** 240 µs for 8 bytes of data.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - UINT8 *u8Buffer: Buffer where data is located.
  - UINT16 u16SizeInBytes: Size of the designated buffer (in bytes). The size must be strictly greater than 0. Setting the size to 0 will cause an unexpected behavior
  - UINT8 u8Remainder: Initial remainder. This argument is useful when the function is used recursively. It must equal "0" if there is no previous data.
- **Returns:**
  - UINT8 u8NewCRC: New calculated CRC8.

3.2.35 UINT16 TPMS_CRC16(UINT8 *u8Buffer, UINT16 u16MByteSize, UINT16 u16Remainder)

- **Description:** Calculates a CRC16 on a portion of the designated memory area by using a look-up table. Polynomial used is $1021 (standard for CRC16-CCITT).
- **Stack size:** 13 bytes
- **Approx. Duration:** 226 µs for 8 bytes.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - UINT8 *u8Buffer: Buffer where data is located.
  - UINT16 u16MByteSize: Size of the designated buffer (in bytes)
  - UINT16 u16Remainder: Initial remainder.
- **Returns:**
  - UINT16 u16NewCRC: New calculated CRC16.

3.2.36 UINT16 TPMS_SQUARE_ROOT(UINT16 u16Process)

- **Description:** Calculates a two-digit remainder of (square root * 10) using a fast algorithm.
- **Stack size:** 49 bytes
- **Approx. Duration:** 365 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - UINT16 u16Process: The number from which to get the square root from.
- **Returns:**
  - UINT16 Root of the number * 10.

Note: *The result may include an error due to truncation in the calculation.*

3.2.37 void TPMS_READ_ID(UINT8 *u8Code)

- **Description:** Copies the device's UniqueID and firmware version stored in firmware flash to RAM.
- **Stack size:** 2 bytes
- **Approx. Duration:** 32 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - UINT8 *u8Code: RAM location where data will be copied. Table 26 describes the format of the 6-bytes returned.

Table 26. u8Code format

Index	Description
0	Firmware version.
1	Derivative descriptor.
2:5	32-bit UniqueID.

- **Returns:**
  – void

3.2.38 void TPMS_LF_ENABLE(UINT8 u8Switch)

- **Description:** Enables/disables the LFR module.
- **Stack size:** 5 bytes
- **Approx. Duration:** 32 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.
- **Resources:** LFR
- **Input Parameters:**
  – UINT8 u8Switch: Enable (non-zero) or disable (zero) LFR.
- **Returns:**
  – void

3.2.39 UINT8 TPMS_LF_READ_DATA(UINT8 *u8Buffer, UINT8 u8Count)

- **Description:** Once the user has configured and enabled the LFR, it is customary to go into a low-power state mode and wait for a datagram. After the first byte of an incoming datagram is successfully received, this function should be called immediately; It will receive the complete datagram and place it in RAM. Be careful to call the function upon reception of the first data byte (LFDRF flag) and not upon detection of the ID (LFIDF flag) in case the LFIDIE is enabled. This function assumes that the LFR module is configured accordingly for a Manchester reception; that the module's interrupts are enabled; and that the first byte has already been received and is waiting in the LFR received buffer. While waiting for the next byte, this function goes into STOP4. If the byte, for an unexpected reason, is not received, this function has a built-in time-out: After five continuous non-LFR interrupts, the function will assume a failed LFR reception and exit. In order to leave the routine as soon as possible after reception of all the data bytes it is recommended to enable the LF error interrupt (LFERIE). In summary, the two necessary interrupts to be enabled are LFDRIE and LFERIE.
- **Stack size:** 7 bytes
- **Approx. Duration:** Data dependent; ~2 ms per byte.
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the LFR interrupt to wake up from Stop mode. It does not await any other interrupts and should not be affected by them.
- **Resources:** LFR
- **Input Parameters:**
  – UINT8 *u8Buffer: RAM Buffer where data will be placed.

  – UINT8 u8Count: Number of bytes expected.
- **Returns:**
  – UINT8 u8BytesReceived: Actual number of bytes received.

Warning: *This function requires ~24 µs from the moment it is called to the moment the first byte is copied into the RAM buffer. The user must consider this time when designing their firmware.*

3.2.40 UINT8 TPMS_WIRE_AND_ADC_CHECK(UINT8 u8TestMask)

- **Description:** This function will check if there is any bonding wire failure between the embedded core and the P-cell; or between the core and the g-cell. It will also perform an optional a g-cell self-test, and/or an ADC test. The latter will consist on taking two reference measurements (ground and Vdd) using internal channels and comparing them with the expected results. The optional g-cell self-test sends a self-test signal and verifies that the g-cell deflects as expected. It can only be called when the device is in parking or static mode. When configuring for a P-cell or g-cell wire check, Interrupts must be enabled before calling this routine. In case of no issues found, "0" will be returned, else it will set status flags as follows:
  – On P-cell wire-bond error, sets "pressure error" flag.
  – On g-cell wire-bond or self-test error, sets "acceleration error" flag.
  – On ADC error, sets the "ADCERR" flag.
- **Stack size:** 36 bytes
- **Approx. Duration:** 14,926 µs (all checks), 106 µs (ADC only), 3,089 µs (P-cell only), 3,087 µs (X-Axis I wire-bond only), 3,089 µs (Z-Axis 1 wire-bond only), 5622 µs (g-cell self-test only).
- **Power Management:** This function requires the core to be configured for STOP4 mode and running at full bus speed.
- **Interrupt Management:** This function utilizes the ADC interrupt to wake up from Stop mode.
- **Resources:** ADC, SMI, internal bond wires.
- **Input Parameters:**
  – UINT8 u8TestMask: This variable determines what checks are performed as described by Table 27.

Table 27. u8TestMask format

u8Test Mask Bit	Description
BIT0	Reserved.
BIT1	If set, g-cell Self-test performed. Refer to warning below.
BIT2	If set, P-cell bond-wire check performed.
BIT3	If set, X-axis bond-wire check performed.
BIT4	If set, Z-axis bond-wire check performed.
BIT5:6	Reserved.
BIT7	If set, ADC check performed.

- **Returns:**
  – UINT8 u8Status: Status flags as described in Table 28.

Table 28. u8Status valid values for TPMS_WIRE_AND_ADC_CHECK

u8Test Mask Bit	Description
BIT0:1	Always clear.
BIT2	If set, P-cell bond-wire error detected.
BIT3	If set, X-axis bond-wire error detected.
BIT4	If set, Z-axis bond-wire or self-test error detected.
BIT5:6	Always clear.
BIT7	If set, ADC error detected.

Warning: *The self-test option will only return valid readings when the device is static (i.e. the vehicle is not moving). It is the user's responsibility to ensure it is only called while in parking or static mode.*

3.2.41 void TPMS_FLASH_WRITE(UINT16 u16Address, UINT8* u8Buffer, UINT8 u8Size)

- **Description:** This function writes consecutive bytes from a given address in memory to a specified location in FLASH.
- **Stack size:** 15 bytes
- **Approx. Duration:** 1336 μs for 8 bytes of data.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will be affected by interrupts.
- **Resources:** Global RAM locations $0090 - $00CA.
- **Input Parameters:**
  - UINT16 u16Address: Flash starting address.
  - UINT8 *u8Buffer: Source memory address.
  - UINT8 u8Size: Number of data bytes to be written. The number of bytes must be strictly greater than 0. Calling the function with a number of bytes equal to 0 may cause user FLASH memory corruption.
- **Returns:**
  - void

Warning: *This routine will overwrite the contents of RAM locations $0090 - $00CA.*

3.2.42 UINT16 TPMS_FLASH_CHECK(void)

- **Description:** This function calculates the CRC16 checksum for the NXP firmware area (addresses $E000 - FFAD) using the function TPMS_CRC16. It compares it with a pre-calculated stored value and reports if these two values match or not.
- **Stack size:** 17 bytes
- **Approx. Duration:** 214070 μs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.

- **Resources:** N/A
- **Input Parameters:**
  - None.
- **Returns:**
  - UINT16 u16Status: "0" in case the calculated checksum and the stored one are the same, or the calculated checksum in case they are different.

3.2.43 UINT8 TPMS_FLASH_ERASE(UINT16 u16Address)

- **Description:** This function erases 1 page (512 bytes) of flash at a time.
- **Stack size:** 11 bytes
- **Approx. Duration:** 22744 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It may be affected by interrupts.
- **Resources:** Global RAM locations $0090 - $00CA.
- **Input Parameters:**
  - UINT16 u16Address: any given address. The whole page where this address resides will be erased (i.e. if u16Address = $D234, the contents of addresses $D200 - $D3FF will be erased).
- **Returns:**
  - Zero if the page was erased successfully; else, one.

Warning: *This routine will overwrite the contents of RAM locations $0090- $00CA.*

3.2.44 UINT8 TPMS_FLASH_PROTECTION(UINT8 u8Range, UINT16 u16Key)

- **Description:** This function enables flash protection for a custom memory block. After its execution, both TPMS_FLASH_WRITE and TPMS_FLASH_ERASE will be unable to perform any operation in the protected blocks, but shall be operational in unprotected areas.
- **Stack size:** 11 bytes
- **Approx. Duration:** 736 µs if protecting; 20.4 µs for a failed attempt.
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will be affected by interrupts.
- **Resources:** Global RAM locations $0090 - $00CA.
- **Input Parameters:**
  - UINT8 u8Range: Byte determining the first address of protection, using the NXP FPROT convention. Refer to product spec for a description of this register.
  - UINT16 u16Key: Due to the irreversible status after the execution of this routine, this argument is used as a fail-safe to guarantee desired execution of the function. Only when u16Key is equal to the least significant word of the UniqueID will this function execute successfully.
- **Returns:** UINT8 u8Status: according to table below.

Table 29. Possible status values for TPMS_FLASH_PROTECTION

Return Value	Description
$0	u16Key matches lower 16-bits of UniqueID; protection was disabled and now is enabled.
$1	u16Key does not match lower 16-bits of UniqueID; protection was disabled and continues to be so.
$2	u16Key matches lower 16-bits of UniqueID; protection was already enabled.
$3	u16Key does not match lower 16-bits of UniqueID; protection was already enabled.
$4	u16Key matches lower 16-bits of UniqueID; protection was disabled and continues to be so (Failed writing protection).
$5 - $FF	Reserved.

Warning: *This routine can only be executed once. After successful execution of this routine, TPMS_FLASH_WRITE and TPMS_FLASH_ERASE will be permanently disabled.*

3.2.45 void TPMS_MULT_SIGN_INT16(INT16 i16Mult1, INT16 i16Mult2, INT32* pi32Result)

- **Description:** This function will multiply two signed 16-bit numbers together.
- **Stack size:** 17 bytes
- **Approx. Duration:** 60 µs
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It should not be affected by interrupts.
- **Resources:** N/A
- **Input Parameters:**
  - INT16 i16Mult1: First multiplier.
  - INT16 i16Mult2: Second multiplier.
  - INT32* pi32Result: Pointer to a 32-bit variable where the result will be stored.
- **Returns:**
  - void

3.2.46 UINT16 TPMS_WAVG(UINT8 u8PNew, UINT16 u16POld, UINT8 u8Avg)

- **Description:** This subroutine calculates a new weighed average value for a given new and old measurement readings by using the following equation:

$$= \left(\left(\mathrm{u16POld} \times \left(\mathrm{u8AVG} - 1\right) + \mathrm{u8PNew}\right) / \left(\mathrm{u8Avg}\right)\right) \quad (1)$$

- **Stack size:** 12 bytes
- **Approx. Duration:** 38 µs (average of 2), 43 µs (average of 4), 48 µs (average of 8), 53 µs (average of 16), 56 µs (average of 32).
- **Power Management:** This function executes entirely in RUN mode.
- **Interrupt Management:** This function does not await any interrupts. It will not be affected by interrupts.

- **Resources:** N/A
- **Input Parameters:**
  - UINT8 u8PNew: new value to include in average.
  - UINT16 u16Pold: Old average.
  - UINT8 u8Avg: Weight of the average. This value can be 2, 4, 8, 16, 32; any other value will return an incorrect response.
- **Returns:**
  - UINT16 u8NewAverage: resulting weighed average of both old average and the new value (refer to Example 1.

Restriction: *This function is not correctly implemented. The function exists and the user may execute the function but the function does not return the expected data described above. The returned, incorrect data must not be used by the application and the user should avoid using this function.*

3.2.47 void TPMS_RDE_ADJUST_PRESSURE(UINT16* pu16UUMA, T_RDE* ptRDEValues)

- **Description:** This routine's functionality has been removed, but its prototype is still callable for backward compatibility. It always returns CLEAR, and does not affect any passed argument.
- **Stack size:** 4 bytes.
- **Approx. Duration:** 9 µs.
- **Power Management: Interrupt Management:** This function does not await any interrupts. It should not be affected by interrupts.
- **Resources:** Core
- **Input Parameters:**
  - UINT16 *pu16UUMA: Pointer to Universal Uncompensated Measurement Array (as described in Section 2.3). No values are affected.
  - T_RDE* ptRDEValues: Pointer to an array of elements. For more information on the RDE structure, refer to Section 2.5.
- **Returns:** UINT8 u8Status:Always CLEAR.

4 Legal information

4.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

4.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

4.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Tab. 1. Global variables and their locations 4
Tab. 2. TPMS_INTERRUPT_FLAG format and trigger conditions 4
Tab. 3. Error status fields 6
Tab. 4. Universal uncompensated measurement array 7
Tab. 5. Customer-configurable TPMS7 LF registers with SENS = 1 10
Tab. 6. Customer-configurable TMPS and LF registers with SENS = 2 10
Tab. 7. FXTH87xx11 and FXTH87xx12's firmware function jump table 11
Tab. 8. Valid output conditions for TPMS_READ_VOLTAGE 13
Tab. 9. Valid output conditions for TPMS_COMP_VOLTAGE 14
Tab. 10. Valid output conditions for TPMS_READ_TEMPERATURE 15
Tab. 11. Valid output conditions for TPMS_COMP_TEMPERATURE 16
Tab. 12. Valid output conditions for TPMS_READ_PRESSURE 17
Tab. 13. Valid output conditions for TPMS_COMP_PRESSURE 18
Tab. 14. u8DynamicOffset valid values 19
Tab. 15. Valid output conditions for TPMS_READ_ACCELERATION_X 19
Tab. 16. Valid output conditions for TPMS_COMP_ACCELERATION_X 21
Tab. 17. Valid output conditions for TPMS_READ_ACCELERATION_Z 22
Tab. 18. Valid output conditions for TPMS_COMP_ACCELERATION_Z 23
Tab. 19. Valid output conditions for TPMS_READ_ACCELERATION_XZ 25
Tab. 20. Valid output conditions for TPMS_COMP_ACCELERATION_XZ 26
Tab. 21. Valid output conditions for TPMS_READ_V0 and TPMS_READ_V1 27
Tab. 22. u16RFParam array format 32
Tab. 23. Description of element 0 in the u16RFParam array 32
Tab. 24. *pu8PowerManagement format 33
Tab. 25. pu8PowerManagement[0] format 33
Tab. 26. u8Code format 37
Tab. 27. u8TestMask format 38
Tab. 28. u8Status valid values for TPMS_WIRE_AND_ADC_CHECK 39
Tab. 29. Possible status values for TPMS_FLASH_PROTECTION 41

Figures

Fig. 1. Measurement signal range definitions 6
Fig. 2. Description of the physical layer on the FXTH87xx11 and FXTH87xx12 simulated SPI interface 9

Contents

1 Introduction 4
2 Globals and formats 4
2.1 Global variables 4
2.1.1 TPMS_INTERRUPT_FLAG 4
2.1.2 TPMS_CONT_ACCEL_GLOBAL_VARIABLE 5
2.2 Measurement error format 5
2.2.1 Definition of signal ranges 5
2.2.2 Error status format 6
2.3 Universal uncompensated measurement array (UUMA) format 7
2.4 Simulated SPI interface signal format 8
2.5 Rapid decompression event array (T_RDE) format 9
2.6 LFR registers initialized by firmware 9
3 Firmware functions 11
3.1 Firmware jump table 11
3.2 Function description 12
3.2.1 void TPMS_RESET(void) 13
3.2.2 UINT8 TPMS_READ_VOLTAGE(UINT16 *u16UUMA) 13
3.2.3 UINT8 TPMS_COMP_VOLTAGE(UINT8 *u8CompVoltage, *UINT16 u16UUMA) 14
3.2.4 UINT8 TPMS_READ_TEMPERATURE(UINT16 *u16UUMA) 14
3.2.5 UINT8 TPMS_COMP_TEMPERATURE(UINT8 *u8Temp, UINT16 *u16UUMA) 15
3.2.6 UINT8 TPMS_READ_PRESSURE(UINT16 *u16UUMA, UINT8 u8Avg) 16
3.2.7 UINT8 TPMS_COMP_PRESSURE(UINT16 *u16CompPressure, UINT16 *u16UUMA) 17
3.2.8 UINT8 TPMS_READ_ACCELERATION_X(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffset) 18
3.2.9 UINT8 TPMS_READ_DYNAMIC_ACCEL_X(UINT8 u8Filter, UINT8* u8Offset, UINT16* u16UUMA) 20
3.2.10 UINT8 TPMS_COMP_ACCELERATION_X(UINT16 *u16CompAccelX, UINT16* u16UUMA) 20
3.2.11 UINT8 TPMS_READ_ACCELERATION_Z(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffset) 21
3.2.12 UINT8 TPMS_READ_DYNAMIC_ACCEL_Z(UINT8 u8Filter, UINT8* u8Offset, UINT16* u16UUMA) 22
3.2.13 UINT8 TPMS_COMP_ACCELERATION_Z(UINT16 *u16CompAccel, UINT16* u16UUMA) 23
3.2.14 UINT8 TPMS_READ_ACCELERATION_XZ(UINT16 *u16UUMA, UINT8 u8Avg, UINT8 u8FiltSelect, UINT8 u8DynamicOffsetX, UINT8 u8DynamicOffsetZ) 24
3.2.15 UINT8 TPMS_READ_DYNAMIC_ACCEL_XZ(UINT8 u8Filter, UINT8* u8OffsetX, UINT8* u8OffsetZ, UINT16* u16UUMA) 25
3.2.16 UINT8 TPMS_COMP_ACCELERATION_XZ(UINT16 *u16CompAccel, UINT16* u16UUMA) 26
3.2.17 UINT8 TPMS_READ_V0(UINT16 *u16Result, UINT8 u8Avg) 27
3.2.18 UINT8 TPMS_READ_V1(UINT16 *u16Result, UINT8 u8Avg) 27
3.2.19 UINT8 TPMS_LFOCAL(void) 28
3.2.20 UINT8 TPMS_MFOCAL(void) 28
3.2.21 void TPMS_RF_ENABLE(UINT8 u8Switch) 29
3.2.22 void TPMS_RF_RESET(void) 29
3.2.23 void TPMS_RF_READ_DATA(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer) 30
3.2.24 void TPMS_RF_READ_DATA_REVERSE(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer) 30
3.2.25 void TPMS_RF_WRITE_DATA(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer) 31
3.2.26 void TPMS_RF_WRITE_DATA_REVERSE(UINT8 u8Size, UINT8 *u8RAMBuffer, UINT8 u8RFMBuffer) 31
3.2.27 void TPMS_RF_CONFIG_DATA(UINT16 *u16RFParam) 31
3.2.28 void TPMS_RF_SET_TX(UINT8 u8BufferSize) 32
3.2.29 void TPMS_RF_DYNAMIC_POWER(UINT8 u8CompT, UINT8 u8CompV, UINT8* pu8PowerManagement) 33
3.2.30 void TPMS_MSG_INIT(void) 34
3.2.31 UINT8 TPMS_MSG_READ(void) 34
3.2.32 UINT8 TPMS_MSG_WRITE(UINT8 u8SendByte) 34
3.2.33 UINT8 TPMS_CHECKSUM_XOR(UINT8 *u8Buffer, UINT8 u8Size, UINT8 u8Checksum) 35
3.2.34 UINT8 TPMS_CRC8(UINT8 *u8Buffer, UINT16 u16SizeInBytes, UINT8 u8Remainder) 35
3.2.35 UINT16 TPMS_CRC16(UINT8 *u8Buffer, UINT16 u16MByteSize, UINT16 u16Remainder) 36
3.2.36 UINT16 TPMS_SQUARE_ROOT(UINT16 u16Process) 36
3.2.37 void TPMS_READ_ID(UINT8 *u8Code) 36
3.2.38 void TPMS_LF_ENABLE(UINT8 u8Switch) 37
3.2.39 UINT8 TPMS_LF_READ_DATA(UINT8 *u8Buffer, UINT8 u8Count) 37
3.2.40 UINT8 TPMS_WIRE_AND_ADC_CHECK(UINT8 u8TestMask) 38

3.2.41 void TPMS_FLASH_WRITE(UINT16 u16Address, UINT8* u8Buffer, UINT8 u8Size) 39
3.2.42 UINT16 TPMS_FLASH_CHECK(void) 39
3.2.43 UINT8 TPMS_FLASH_ERASE(UINT16 u16Address) 40
3.2.44 UINT8 TPMS_FLASH_PROTECTION(UINT8 u8Range, UINT16 u16Key) 40
3.2.45 void TPMS_MULT_SIGN_INT16(INT16 i16Mult1, INT16 i16Mult2, INT32* pi32Result) 41
3.2.46 UINT16 TPMS_WAVG(UINT8 u8PNew, UINT16 u16POld, UINT8 u8Avg) 41
3.2.47 void TPMS_RDE_ADJUST_PRESSURE(UINT16* pu16UUMA, T_RDE* ptRDEValues) 42
4 Legal information 43